Hash-funktiot

LuK-tutkielma
Markus Rautell
2426501
Matemaattisten tieteiden laitos
Oulun yliopisto
Kevät 2020

Sisältö

Johdanto

Tämän tutkielman aiheena ovat hash-funktiot. Tutkielmassa perehdytään hash-funktioiden yleisluonteeseen, tarkastellaan SHA-1 toimintaa ja arvioidaan hash-funktioiden käyttökelpoisuutta syntymäpäivähyökkäyksien avulla. Käytännössä perehdytään hash-funktioilta edellytettäviin ominaisuuksiin, siihen miten niille asetetut vaatimukset täyttyvät, tarkastellaan ykstyiskohtaisesti SHA-1 algoritmia ja sen eri vaiheita ja arvioidaan hash-funktioiden käytön turvallisuutta todennäköisyyslaskentaa hyödyntämällä.

Tutkielmassa aloitetaan aiheeseen perehtyminen käsittelemällä hash-funktioita hyvin yleisellä tasolla. Käydään läpi missä kaikkialla hash-funktiot esiintyvät, millaisia laskennallisia ominaisuuksia niihin liittyy ja esitellään yksi käytännön esimerkki. Kun hash-funktion konsepti on hallussa, käännetään katse SHA-1 algoritmiin. Tässä käydään läpi hieman historiaa ja algoritmin toimintaperiaatetta. Tämän jälkeen pureudutaan syvemmin algoritmiin ja sen käyttämiin operaatiohin. Lopuksi esitellään syntymäpäivähyökkäykset ja havainnollistetaan, että millaisia menetelmiä käytetään hash-funktioden turvallisuuden arvioimiseksi. Tässä osassa havainnollistetaan selkeästi syyt, miksi hash-funktioilta vaaditaan tiettyjä ominaisuuksia, mitä tutkielmassa aikaisemmin on esitetty.

Tutkielmassa on käytetty pääasiassa teosta [1].

1 Hash-funktiot

1.1 Johdanto

Hash-funktio on eräänlainen tiivistefunktio, joka ottaa syötteenä merkkijonon ja tuottaa siitä tietyn mittaisen tiivisteen. Tietotekniikassa hash-funktioita käytetään esimerkiksi alkioiden taulukoinnissa sekä salasanojen ja tiedostojen eheyden tarkistamisessa. Salasanojen säilöminen tietokannassa sellaisenaan on suuri tietoturvariski. Jos tietokantaan päästäisiin murtautumaan, salasanat joutuisivat vääriin käsiin. Tämän takia on turvallisempaa säilöä vain salasanoista generoituja tiivistemerkkijonoja. Tällöin voidaan kirjautumishetkellä verrata käyttäjän syöttämästä salasanasta tuotettua tiivistettä siihen tiivisteeseen, mikä on alunperin tallennettu tietokantaan. Jos tiivisteet ovat samat, kirjautuminen onnistuu.

Tämä tutkielma keskittyy hash-funktioiden toimintaan matemaattisesta näkökulmasta ja niiden käyttöön kryptografiassa. Tutkielmassa käsitellään hash-funktioiden perusominaisuuksia ja niihin kohdistuvia hyökkäyksiä. Hash-funktio on monen kryptografisen algoritmin peruskomponentti ja sen täyttäessä tietyt yksisuuntaisuus ominaisuudet, sitä voidaan käyttää tehostamaan monia algoritmeja.

1.2 Kryptografia ja hash-funktiot

Kryptografinen hash-funktio h ottaa syötteenä mielivaltaisen pitkän viestin ja tuottaa siitä tietyn mittaisen merkkijonon. Hash-funktion tulee täyttää seuraavat ominaisuudet:

1. Syötetystä viestistä m voidaan tuottaa tiiviste $h(m)$ hyvin nopeasti.
2. Jos tiedetään tiiviste y on mahdotonta selvittää viesti m' siten, että $h(m') = y$. Toisin sanoen h on yksisuuntainen tai alkukuva resistentti.
3. On laskennallisesti mahdotonta selvittää viestit m_1 ja m_2 tiedon $h(m_1) = h(m_2)$ avulla (tässä tapauksessa funktion h sanotaan olevan voimakkaasti törmäysvapaa).

Näistä kolmesta ominaisuus 3 on vaikein täyttää, sillä on otettava huomioon, että mahdollisten viestien m joukko on paljon suurempi kuin mahdollisten tiivisteiden joukko $h(m)$. Tämän takia aina pitäisi olla olemassa paljon esimerkkejä tapauksista, joissa tapahtuu törmäys eli on olemassa kaksi erilaista viestiä m_1 ja m_2, joista syntyy tiivisteet $h(m_1)$ ja $h(m_2)$, jotka ovat samat ($m_1 \neq m_2 \Rightarrow h(m_1) = h(m_2)$). Ominaisuus 3 tarkoittaa, että pitäisi olla vaikeaa löytää näitä esimerkkejä törmäyksistä. Käytännön tasolla toisinaan

kuitenkin riittää, että h on vain heikosti törmäysvapaa. Tämä tarkoittaa sitä, että annetulle alkukuvalle x on laskennallisesti mahdotonta löytää sellainen x', että $h(x) = h(x')$. Tätä ominaisuutta kutsutaan toiseksi alkukuvaresistentiksi.

1.3 Digitaalinen allekirjoitus

Yksi hash-funktioiden pääasiallinen käyttö kohde on digitaalinen allekirjoitus. Yleensä digitaalisen allekirjoituksen pituus on vähintään yhtä pitkä kuin allekirjoitettava dokumentti, joten on paljon tehokkaampaa muodostaa allekirjoitus dokumentin tiivisteestä.

Tässä tapauksessa hash-funktio on julkinen. Henkilö A muodostaa viestistä m tiivisteen $h(m)$. Tiiviste $h(m)$ on merkittävästi pienempi kuin viesti m ja allekirjoitus voidaan tehdä nopeammin. Henkilö A muodostaa allekirjoituksen $sig(h(m))$ ja käyttää sitä viestin allekirjoituksena. Henkilö A lähettää parin $(m, sig(h(m)))$ henkilölle B. Tämän menetelmän etuna ovat allekirjoituksen nopea tuottaminen ja se, että tiedonsiirrossa ja varastoinnissa tarvitaan käyttöön vähemmän resursseja kuin tapauksessa, jossa allekirjoitetaan koko alkuperäinen viesti.

Onko tämä metodi turvallinen? Nyt henkilö C saa käsiinsä henkilölle B tarkoitetun parin $(m, sig(h(m)))$ ja haluaa käyttää henkilön A allekirjoitusta omassa viestissään m'. Tällöin hän tarvitsee törmäyksen $sig(h(m)) = sig(h(m'))$ eli $h(m) = h(m')$. Jos käytetty hash-funktio on yksisuuntainen, henkilön C on vaikea löytää tälläinen m'. Koska hash-funktiolta vaaditaan törmäysvapaata ominaisuutta on epätodennäköistä, että henkilö C löytää viestin $m' \neq m$, johon sopisi henkilön A allekirjoitus $sig(h(m))$.

1.4 Tiedon eheys

Hash-funktioita voidaan käyttää myös tiedon eheyden tarkistamiseen. Kysymys tiedon eheydestä tulee esille yleensä kahdessa skenaariossa. Ensimmäisessä dataa siirretään käyttäjältä toiselle ja meluisa kommunikaatiokanava aiheuttaa virheitä siihen. Toisessa skenaariossa kolmas osapuoli kaappaa viestin ja muuttaa sitä, ennen kuin se pääsee vastaanottajalle. Molemmissa tapauksissa data vioittuu.

Hash-funktion tehtävänä on todentaa vastaanottajalle, että onko data saapunut perille alkuperäisessä muodossa. Lähetetään pari $(m, h(m))$, jossa m on alkuperäinen viesti ja $h(m)$ viestistä tehty tiiviste, kommunikaatiokanavan yli ja se vastaanotetaan muodossa (M, H), jossa $M = m$ ja $H = h(m)$. Tarkastaakseen, ettei datassa ole virheitä, vastaanottaja laskee $h(M)$ tuloksen ja vertaa sitä vastaanotettuun komponenttiin H. Jos data on vioittunut

matkalla vastaanottajalle, on hyvin todennäköistä, että $h(m) \neq H$. Tämä johtuu hash-funktion h törmäysvapaasta ominaisuudesta.

1.5 Diskreetin logaritmin ongelma

Seuraava esimerkki hash-funktiosta täyttää ominaisuuksien 2 ja 3 vaatimukset, mutta on aivan liian hidas oikeaan käyttöön. Tästä huolimatta se demonstroi hyvin hash-funktion perusajatusta.

Aloitetaan valitsemalla suuri alkuluku p siten, että $q = (p-1)/2$ on myös alkuluku. Seuraavaksi valitaan p:lle kaksi primitiivijuurta α ja β. Koska α on primitiivijuuri, on olemassa sellainen a, että $\alpha^a \equiv \beta \pmod{p}$. Oletetaan, ettei tälläistä lukua a tiedetä, koska luvun a selvittäminen vaatii diskreetin logaritmin ongelman ratkaisemista, mikä oletetaan vaikeaksi.

Hash-funktio h kuvaa kokonaisluvut mod q^2 kokonaisluvuiksi mod p. Tästä johtuen h:n muodostamassa tiivisteessä on noin puolet vähemmän bittejä kuin alkuperäisessä viestissä. Tämä ei ole niin dramaattinen koon pieneneminen kuin yleensä käytännössä tarvitaan, mutta se sopii nyt tähän tarkoitukseen.

Olkoon $m = x_0 + x_1 q$, jolle pätee $0 \leq x_0, x_1 \leq q-1$. Määritellään

$$h(m) \equiv \alpha^{x_0} \beta^{x_1} \pmod{p}.$$

Seuraavaksi todistetaan, että funktio h on voimakkaasti törmäysvapaa.

Lemma 1.1. *Jos tiedetään viestit $m \neq m'$, joille pätee $h(m) = h(m')$. Tällöin voimme selvittää diskreetin logaritmin $a = L_\alpha(\beta)$.*

Todistus. Olkoon $m = x_0 + x_1 q$ ja $m' = x_0' + x_1' q$. Oletetaan, että

$$\alpha^{x_0} \beta^{x_1} \equiv \alpha^{x_0'} \beta^{x_1'} \pmod{p}.$$

Tiedetään, että $\beta \equiv \alpha^a \pmod{p}$. Tällöin saadaan yhtälö muotoon

$$\alpha^{a(x_1 - x_1') - (x_0' - x_0)} \equiv 1 \pmod{p}.$$

Koska α on primitiivijuuri mod p tiedetään, että $\alpha^k \equiv 1 \pmod{p}$ jos ja vain jos $k \equiv 0 \pmod{p-1}$. Tässä tapauksessa se tarkoittaa, että

$$a(x_1 - x_1') \equiv x_0' - x_0 \pmod{p-1}.$$

Olkoon $d = syt(x_1 - x_1', p-1)$. Tällöin on tasan d mahdollista ratkaisua edellä esitettyyn kongruenssiyhtälöön ja ne voidaan löytää nopeasti. Luvun p määrittelyn takia $p-1$ ainoat tekijät ovat $1, 2, q, p-1$. Nyt $0 \leq x_0, x_1 \leq q-1$ ja siitä seuraa, että $-(q-1) \leq x_1 - x_1' \leq q-1$. Tällöin, jos $x_1 - x_1' \neq 0$, niin kyseessä on nollasta eroava d:n monikerta, jonka itseisarvo on pienempää kuin q. Tämä tarkoittaa, että $d \neq q, p-1$, joten $d = 1$ tai $d = 2$. Tällöin on enintään kaksi mahdollista arvoa luvulle a. Lasketaan α^a molemmilla arvoilla, joista vain toinen tuottaa vastaukseksi luvun β. Tämän takia olemme löytäneet halutun luvun a.

Toisaalta, jos $x_1 - x_1' = 0$ siitä seuraa, että $x_0' - x_0 \equiv 0 \pmod{p-1}$. Koska $-(q-1) \leq x_1 - x_1' \leq q-1$, on oltava $x_0' = x_0$. Tällöin $m = m'$, mikä on ristiriidassa alkuperäisen väitteen kanssa. □

Jos hash-funktio ei olisi voimakkaasti törmäysvapaa, niin Lemman 1.1 nojalla diskreetin logaritmin ongelma olisi ratkaistavissa. Näin ollen on osoitettu, että hash-funktio on voimakkaasti törmäysvapaa, koska oletetaan, että diskreetin logaritmin ongelmaa ei voida ratkaista.

Nyt on helppo osoittaa, että h on alkukuvaresistentti. Olkoon meillä algoritmi g, joka ottaa syötteenä tiivisteen y ja nopeasti löytää viestin m, jolle pätee $h(m) = y$. Tässä tapauksessa on helppoa löytää $m_1 \neq m_2$ siten, että $h(m) = h(m')$: Valitaan satunnainen m ja lasketaan $y = h(m)$, minkä jälkeen lasketaan $g(y)$. Koska h kuvaa q^2 muotoa olevat viestit $p-1 = 2q$ muotoiseksi tiivisteeksi, on olemassa monia viestejä m', joille $h(m) = h(m')$. Tämän takia ei ole kovin todennäköistä, että $m' = m$. Jos näin on, kokeillaan uudella satunnaisella viestillä m. Pian pitäisi löytyä sellainen törmäys, että viestit $m_1 \neq m_2$, joille pätee $h(m_1) = h(m_2)$. Edeltävä todistus näyttää, että voimme siten ratkaista diskreetin logaritmin ongelman. Tämän takia on epätodennäköistä, että kuvatunlaista funktiota g olisi olemassa.

1.6 Esimerkki hash-funktion toiminnasta

Seuraavaksi esitellään yksinkertainen hash-funktio, jolla on samat perusominaisuudet kuin hash-funktioilla, jotka ovat päivittäisessä käytössä. Esiteltävä hash-funktio ei kuitenkaan ole kovin vahva, joten sitä ei tule käyttää missään systeemeissä.

Aloitetaan viestillä m, jolla on mielivaltainen pituus L. Voimme pilkkoa viestin m n-bittisiin lohkoihin, kun n on paljon pienempi luku kuin L. Merkitään näitä n-bittisiä lohkoja m_j:llä, kun $m = [m_1, m_2, m_3, \ldots, m_l]$. Tässä $l = \lceil L/n \rceil$ ja lohko m_l täytetään nollilla, jotta se olisi n-bitin mittainen.

Kirjoitetaan j. lohko m_j vektori muodossa

$$m_j = [m_{j1}, m_{j2}, m_{j3}, \ldots, m_{jn}],$$

missä jokainen m_{ji} on yksi bitti.

Nyt voidaan muodostaa matriisi näistä vektoreista m_j. Tiivisteen pituus tulee olemaan n-bittiä pitkä, joten lasketaan jokaisen sarakkeen i biteille summa mod 2, joka on $h_i = m_{1i} \oplus m_{2i} \oplus m_{3i} \oplus \ldots \oplus m_{li}$. Tämä visualisoidaan seuraavasti:

$$\begin{bmatrix} m_{11} & m_{12} & \ldots & m_{1n} \\ m_{21} & m_{22} & \ldots & m_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{l1} & m_{l2} & \ldots & m_{ln} \end{bmatrix}$$
$$\begin{array}{cccc} \Downarrow & \Downarrow & \Downarrow & \Downarrow \\ \oplus & \oplus & \oplus & \oplus \\ \Downarrow & \Downarrow & \Downarrow & \Downarrow \end{array}$$
$$[\ c_1 \quad c_2 \quad \ldots \quad c_n\ ] = h(m)$$

Tämä hash-funktio pystyy ottamaan syötteenä mielivaltaisen pitkän viestin ja muokkaamaan sen n-bittikseksi tiivisteeksi. Kyseinen hash-funktio ei ole kuitenkaan kryptografisesti turvallinen, sillä on helppo löytää kaksi viestiä, joilla on sama tiiviste.

2 SHA-1

2.1 Johdanto

SHA-1 (The Secure Hash Algorithm) on National Security Agensyn (NSA) kehittämä hash algoritmi. Alkuperäinen versio (SHA tai SHA-0) julkaistiin vuonna 1993 Yhdysvaltojen tietojenkäsittelystandardina (FIPS 180). SHA:ssa oli kuitenkin heikkouksia, jotka myöhemmin NSA paljasti. Tämä johti uusittuun painokseen (FIPS 180-1), joka julkaistiin vuonna 1995. Uusittu dokumentti esitteli parannellun version SHA-1, josta tuli sittemmin National Institute of Standards and Technologyn suosittelema.

Vuodesta 2005 lähtien SHA-1 ei enää ole pidetty turvallisena hyvin rahoitettuja hyökkääjiä vastaan. NIST lopetti virallisesti SHA-1 käytön vuonna 2011 ja kielsi sen käytön digitaalisessa allekirjoituksessa vuonna 2013. Nykyään SHA-1 on kyetty käytännössä murtamaan. Murtamisprosessi on toistaiseksi hyvin kallis ja aikaan vievä, eikä sitä pysty kuka tahansa yksityishenkilö toteuttamaan. Nykypäivänä on julkaistu tehokkaammat algoritmit SHA-2

ja SHA-3, joihin ei ole kyetty onnistuneesti käyttämään SHA-1 hyökkäystä. Tutkijat kuitenkin pitävät mahdollisena, että lähitulevaisuudessa SHA-2 olisi myös murrettavissa.[2]

SHA-1 tuottaa 160-bitin tiivisteen ja se käyttää iteratiivista menettelyä. Samoin kuin kappaleessa 1.6 alkuperäinen viesti m pilkotaan tietyn pituisiksi lohkoiksi, $m = [m_1, m_2, \ldots, m_l]$, missä viimeinen lohko täydennetään nollilla. Viestilohkot prosessoidaan kierroksittain tiivistefunktiolla h', joka yhdistää käsittelyssä olevan lohkon ja viimeisimmän kierroksen tuotoksen. Tästä johtuen prosessi aloitetaan alkuarvolla X_0 ja määritellään $X_j = h'(X_{j-1}, m_j)$. Lopullinen X_l on viestin tiiviste.

Hash-funktion rakentamisen perustana on luoda hyvä tiivistefunktio. Tämä tiivistefunktio tulisi rakentaa siten, että jokainen syötteenä otettu bitti vaikuttaisi mahdollisimman moneen funktiolla tuotettuun bittiin.

2.2 Alustus

SHA-1 alkaa alkuperäisen viestin täyttämisellä bitillä 1 ja sarjalla 0 bittejä. 0 bittejä lisätään loppuun niin kauan, että uusi muodostunut viesti on 64 bittiä vajaa seuraavaksi korkeimmasta 512 bitin monikerran pituudesta. Seuraavaksi muodostetaan viestin T pituudesta 64 bittinen esitys ja lisätään se jo laajennetun viestin loppuun. Jos viesti on T bittinen, edellinen 64 bitin lisäys luo viestin, joka muodostuu $L = \lfloor T/512 \rfloor + 1$ kappaleesta 512 bittisiä lohkoja. Sitten laajennettu viesti pilkotaan L lohkoon $m_1, m_2, \ldots, m_L$. Hash-algoritmi ottaa lohkot syötteenä käsittelyyn yksitellen.

Esimerkki 2.1. Olkoon meillä 4000 bittinen viesti. Nyt

$$4000 = 7 \times 512 + 416.$$

On siis lisättävä luku 1 ja 31 kappaletta nollia viestin perään, jotta saavutetaan pituus

$$4032 = 8 \times 512 - 64.$$

Koska $4000 = 111110100000_2$, on lisättävä $64 - 12 = 52$ kappaletta nollia binääriluvun perään, jotta saavutetaan 4096 bitin pituus. Tämä pilkotaan kahdeksaan 512 bitin lohkoon.

Hash-funktion kuvauksessa tarvitaan seuraavia operaatioita 32 bittisille merkkijonoille:

1. $X \wedge Y$ = "ja" operaatio biteille, mikä tarkoittaa bittien kertolaskua mod 2.

2. $X \vee Y =$ "tai" operaatio biteille, missä kahdesta bitistä valitaan tulokseen suurempi.

3. $X \oplus Y =$ bittien yhteenlasku mod 2.

4. $\neg X$ muuttaa ykköset nolliksi ja nollat ykkösiksi.

5. $X+Y = X$:n ja Y:n summa mod 2^{32}, missä X ja Y ovat kokonaislukuja mod 2^{32}.

6. $X \hookleftarrow r = X$:n siirtyminen vasemmalle r askelta (ja alkupää siirtyy loppuun).

Tarvitaan myös seuraavia funktioita:

$$f_t(B,C,D) = \begin{cases} (B \wedge C) \vee ((\neg B) \wedge D), & \text{jos } 0 \leq t \leq 19 \\ B \oplus C \oplus D, & \text{jos } 20 \leq t \leq 39 \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D), & \text{jos } 40 \leq t \leq 59 \\ B \oplus C \oplus D, & \text{jos } 60 \leq t \leq 79. \end{cases}$$

Määritellään vakiot $K_0, ..., K_{79}$ seuraavasti:

$$K_t = \begin{cases} 5A827999, & \text{jos } 0 \leq t \leq 19 \\ 6ED9EBA1, & \text{jos } 20 \leq t \leq 39 \\ 8F1BBCDC, & \text{jos } 40 \leq t \leq 59 \\ CA62C1D6, & \text{jos } 60 \leq t \leq 79. \end{cases}$$

Edellä olevat on kirjoitettu *heksadesimaalimuodossa*. Jokainen luku tai kirjain kuvaa 4 bitin merkkijonoa:

$$0 = 0000, 1 = 0001, 2 = 0010, ..., 9 = 1001$$
$$A = 1010, B = 1011, ..., F = 1111.$$

Esimerkiksi A1F $= 10 \times 16^2 + 1 \times 16^1 + 15 = 2591$.

2.3 SHA-1 Algoritmi

1. Aloitetaan viestistä m. Laajennus biteillä, joista mainittiin esimerkkissä 2.1, saadaan viesti y, joka on muotoa $y = m_1||m_2||...||m_L$, missä joikaisessa lohkossa m_i on 512 bittiä.

2. Asetetaan alkuarvot $H_0 = 67452301, H_1 = EFCDAB89$,
$H_2 = 98BADCFE, H_3 = 10325476, H_4 = C3D2E1F0$.

3. Jokaiselle $i = 0; L - 1$ asti, suoritetaan seuraava:

   (a) Kirjoitetaan $m_i = W_0||W_1||...||W_{15}$, missä jokaisessa W_j on 32 bittiä.

   (b) Jokaiselle $t = 16; 79$ asti, olkoon $W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \hookleftarrow 1$

   (c) Olkoon $A = H_0, B = H_1, C = H_2, D = H_3, E = H_4$.

   (d) Jokaiselle $t = 0; 79$ asti, suorita seuraavat vaiheet peräkkäin:
   $T = (A \hookleftarrow 5) + f_t(B, C, D) + E + W_t + K_t, E = D$
   $D = C, C = (B \hookleftarrow 30, B = A, A = T$.

   (e) Olkoon $H_0 = H_0 + A, H_1 = H_1 + B, H_2 = H_2 + C$,
   $H_3 = H_3 + D, H_4 = H_4 + E$.

4. Tuotos $H_0||H_1||H_2||H_3||H_4$. Tämä on 160 bittinen hash-arvo.

Algoritmin ydin on askel 3. Kaikki SHA-1 sisältyvät algoritmit ovat yksinkertaisia ja todella nopeita. Huomaa, että perusmenettely toistetaan niin monta kertaa kuin tarvitaan koko viestin tiivistämiseksi. Tämä iteratiivinen menettely tekee algoritmista todella tehokkaan viestin lukemisen ja käsittelyn kannalta.

Käydään nyt algoritmi läpi. SHA-1 alkaa luomalla alustavan 160 bittisen rekisterin X_0, joka sisältää viisi 32 bitin alirekisteriä H_0, H_1, H_2, H_3, H_4. Nämä alirekisterit on alustettu yllä olevan kohdan 2 mukaisesti. Sen jälkeen, kun lohko m_j on käsitelty, rekisteri X_j on päivitetty rekisterin X_{j+1} tuottamista varten.

Jokainen 512 bitin viestilohko m_j kulkee SHA-1 silmukan läpi. Jokaiselle viestilohkolle, m_j, rekisteri X_j kopioidaan alirekistereihin A, B, C, D, E. Aloitetaan viestilohkosta m_0, joka on paloiteltu ja sekoitettu tuottaen $W_0, ..., W_{79}$. Nämä syötetään neljän kierroksen sekvenssiin, vastaamaan neljää intervallia $0 \leq t \leq 19, 20 \leq t \leq 39, 40 \leq t \leq 59, 60 \leq t \leq 79$. Jokaisella kierroksella syötteenä toimii rekisterin X_0 sen hetkinen arvo ja kyseessä olevan intervallin lohkot W_t, ja suorittaa niillä 20 iteraatiota (laskuri t käy intervallin 20 arvoa läpi). Jokainen iteraatio käyttää kierrosvakiota K_t ja operaatiota $f_t(B, C, D)$, mitkä ovat samat jokaiselle tämän kierroksen iteraatiolle. Yksi toisensa jälkeen, jokainen kierros päivittää alirekisteri (A, B, C, D, E). Neljännen kierroksen tuotos, joka on valmistunut kun $t = 79$, alirekisterit (A, B, C, D, E) lisätään syöte alirekistereihin $(H_0, H_1, H_2, H_3, H_4)$ ja tuotoksena on 160 bittiä, josta tulee uusi rekisteri X_1, joka kopioidaan alirekistereihin (A, B, C, D, E), kun aloitetaan seuraavan viestilohkon m_1 käsittely. Tämän tuotoksen X_1 voidaan katsoa olevan tiivistefunktion h' tuotos, kun sille on annettu syötteenä

X_0 ja m_0. Toisin sanoen, $X_1 = h'(X_0, m_0)$.

Tällä tavoin jatketaan jokaisen 512 bittisen viestilohkon m_j kohdalla, käyttäen edellistä tuotosta X_j syötteenä, kun lasketaan seuraavaa tuotosta X_{j+1}. Siten $X_{j+1} = h'(X_j, m_j)$. Kun on käsitelty kaikki L kappaletta viestilohkoja, lopullinen tuotos on 160 bittinen viestin tiiviste.

Algoritmin kivijalkana toimii operaatioden joukko, jotka suoritetaan alirekistereille kohdassa 3d. Ne suorittavat alirekistereille siirtymisoperaatioita ja mod 2 yhteenlaskua, kuten edellä on kuvattu. SHA-1 käyttää myös monimutkaisia sekoitus operaatioita, jotka esiintyvät kuvauksena f_t ja vakioina K_t.

3 Syntymäpäivähyökkäykset

3.1 Johdanto

Jos huoneessa on 23 henkilöä todennäköisyys sille, että kaksi näistä henkilöistä on syntynyt samana päivänä, on hieman yli 50%. Jos taas huoneessa on 30 henkilöä, todennäköisyys on noin 70%. Täma saattaa kuulostaa yllättävältä ja siksi kyseistä ilmiötä kutsutaan syntymäpäiväparadoksiksi. Tarkastellaan hieman tätä ilmiötä, mutta jätetään syntymävuodet huomiotta ja oletetaan, että jokainen syntymäpäivä on yhtä todennäköinen.

Lasketaan todennäköisyys sille, että 23 henkilöä ovat kaikki syntyneet eri päivinä. Laitetaan 23 henkilöä jonoon. Ensimmäinen henkilö käyttää yhden mahdollisista syntymäpäivistä, joten todennäköisyys sille, että jonossa toisena oleva henkilö on syntynyt eri päivänä kuin ensimmäinen, on $(1 - 1/365)$. Tultaessa kolmannen henkilön kohdalle, syntymäpäivistä on käytetty kaksi, joten todennäköisyys on $(1-2/365)$ sille , että kolmas henkilö on syntynyt eri päivänä kuin kaksi ensimmäistä. Tästä seuraa, että todennäköisyys sille, että kaikki kolme henkilöä ovat syntyneet eripäivinä on $(1-1/365) \times (1-2/365)$. Jatkamalla tällä tavoin todennäköisyys sille, että kaikki 23 henkilöä ovat syntyneet eri päivinä on

$$(1 - \frac{1}{365}) \times (1 - \frac{1}{365}) \times ... \times (1 - \frac{22}{365}) = 0,493.$$

Tästä seuraa, että ainakin kaksi henkilöä jakavat saman syntymäpäivän todennäköisyydellä

$$1 - 0,493 = 0,507.$$

Tarkastellaan nyt 40 henkilön tapausta, jotta laskelmasta tulee intuitiivisesti helpompi hahmottaa. Oletetaan, että jonon 30 ensimmäisellä hen-

kilöllä on eri syntymäpäivät. Nyt on enää valittava 10 viimeistä syntymäpäivää. Koska 30 syntymäpäivää on jo valittu, on noin 10% todennäköisyys ($30/365 = 0,0823$), että satunnaisesti valittu syntymäpäivä tulee olemaan sama kuin joku 30 aikaisemmasta. Tästä johtuen ei ole kovinkaan yllättävää, että löydetään joukosta osuma. Itse asiassa, todennäköisyys on

$$1 - (1 - \frac{1}{365}) \times (1 - \frac{2}{365}) \times ... \times (1 - \frac{39}{365}) = 0,891$$

eli noin 89% sille, että 40 henkilön joukosta löytyy osuma.

3.2 Teoriaa

Tarkastellaan edellä esiteltyä yleisellä tasolla. Olkoon meillä N objektia siten, että N on hyvin suuri. Olkoon henkilöitä r kappaletta ja jokainen heistä valitsee objektin takaisinpano menetelmällä, jotta moni voisi valita saman objektin. Siten

$$P(\text{löytyy osuma}) \approx 1 - e^{-r^2/2N}.$$

On otettava huomioon, että tämä on vain approksimaatio, joka pätee, kun N on hyvin suuri. Pienillä arvoilla N on kannattavaa käyttää kappaleessa 3.1 esiteltyä menetelmää. Valittaessa $r^2/2N = ln\,2$ huomataan, että $r \approx 1,177\sqrt{N}$ ja tällöin

$$\begin{aligned} P(\text{löytyy osuma}) &\approx 1 - e^{-(1,177\sqrt{N})^2/2N} = 1 - e^{-1,385N/2N} \\ &= 1 - e^{-1,385/2} \approx 1 - 0,500241 \approx 0,5 \end{aligned}$$

on 50% todennäköisyys sille, että ainakin kaksi henkilöä valitsee saman objektin.

Yhteenvetona, jos on olemassa N mahdollisuutta ja $\sqrt{N}$ pituinen lista, niin on hyvä todennäköisyys saada osuma. Jos halutaan kasvattaa osuman todennäköisyyttä, voidaan kasvattaa listan pituudeksi $2\sqrt{N}$ tai $5\sqrt{N}$. Pointtina on, että vakio kertaa $\sqrt{N}$ on riittävä pituus.

Esimerkki 3.1. Olkoon meillä 35 rekisterikilpeä, joista jokainen loppuu kolminumeroiseen lukuun. Mikä on todennäköisyys sille, että kaksi rekisterikilmpeä loppuu samaan kolminumeroiseen lukuun? Olkoon $N = 729$, joka on kolminumeroisten yhdistelmien määrä, kun lukua 0 ei esiinny lainkaan, ja $r = 35$, joka on tarkasteltavien rekisterikilpien joukko. Siten

$$\frac{r^2}{2N} = \frac{35^2}{2 \times 729} \approx 0,840$$

ja todennäköisyyden approksimaatio on

$$1 - e^{-0,840} \approx 0,568,$$

joka on enemmän kuin 50%. Painotetaan kuitenkin, että tämä on vain approksimaatio. Laskemalla saatu vastaus on

$$1 - (1 - \frac{1}{729}) \times (1 - \frac{2}{729}) \times ... \times (1 - \frac{34}{729}) \approx 0,564.$$

Mikä olisi sitten todennäköisyys, että yhdessä näistä 35 rekisterikilvestä olisi sama kolminumeroinen luku kuin sinun rekisterikilvessäsi? Jokaisella rekisterikilvellä on $1-1/729$ todennäköisyys olla vastaamatta sinun rekisterikilpeäsi, joten 35 rekisterikilvellä tämä todennäköisyys on $(1-1/729)^{35}$=0,953. Syy siihen, miksi syntymäpäiväparadoksi toimii, johtuu siitä, että ei etsitä paria yhdelle tietylle rekisterikilvelle. Etsitään vastaavuutta minkä tahansa kahden rekisterikilven välillä tarkastellusta joukosta. Tämän takia osumien mahdollisuuksia on paljon enemmän.

Näiden oivallusten soveltaminen salauksessa vaatii hieman erilaista järjestelyä. Olkoon kaksi huonetta, joissa molemmissa on 20 henkilöä. Mikä on todennäköisyys, että eräällä henkilöllä ensimmäisessä huoneessa on sama syntymäpäivä kuin eräällä henkilöllä toisessa huoneessa? Yleisemmin, Olkoon N objektia ja kaksi joukkoa, joissa on molemmissa r henkilöä. Jokainen henkilö kummastakin joukosta valitsee objektin takaisinpanomenetelmällä. Mikä on todennäköisyys, että eräs henkilö ensimmäisestä joukosta valitsee saman objektin kuin eräs toisesta joukosta? Jos $\lambda = r^2/N$, niin todennäköisyys osumalle on $1 - e^{-\lambda}$. Todennäköisyys i osumalle on $\lambda^i e^{-\lambda}/i!$.

Taas, jos on olemassa N mahdollisuutta ja kaksi $\sqrt{N}$ pituista listaa, niin on hyvä todennäköisyys saada osuma. Jos halutaan kasvattaa osuman todennäköisyyttä, voidaan kasvattaa listojen pituudeksi $2\sqrt{N}$ tai $5\sqrt{N}$. Pointtina on, että vakio kertaa $\sqrt{N}$ on riittävä pituus.

Esimerkki 3.2. Jos on $N = 365$, $\sqrt{N} \approx 19,1$, joten valitaan $r = 20$, siten

$$\lambda = 20^2/365 = 1,096.$$

Koska $1 - e^{-\lambda} = 0,666$, on noin 66,6% todennäköisyys, että yhdellä henkilöllä ensimmäisessä 20 henkilön joukossa on sama syntymäpäivä kuin yhdellä toisessa 20 henkilön joukossa.

Kasvatetaan listojen kokoa ja tutkintaa sama tilanne, kun listojen pituudeksi määritetään noin $2\sqrt{N}$. Nyt $N = 365$ ja r=40. Tällöin

$$\lambda = 40^2/365 = 4,384.$$

Nyt yhden osuman todennäköisyydeksi saadaan $1 - e^{-4,384} = 0,988$ eli noin 98,8%.

3.3 Hyödyntäminen hash-funktioihin

Syntymäpäivähyökkäystä voidaan käyttää hyödyksi hash-funktion törmäyksien löytämisessä, jos hash-funktion tuotos ei ole tarpeeksi suuri. Olkoon h n-bittinen hash-funktio. Tällöin on olemassa $N = 2^n$ mahdollista tuotosta. Määritetään lista $h(x)$ noin $r = \sqrt{N} = 2^{n/2}$ satunnaisella luvulla x. Tällöin meillä on tilanne, jossa on $r \approx \sqrt{N}$ "henkilöä", joilla on N mahdollista "syntymäpäivää", joten on hyvä todennäköisyys, että on kaksi arvo x_1 ja x_2, joilla on sama hash-arvo. Jos kasvatamme listaa esimerkiksi niin, että x saa $r = 10 \times 2^{n/2}$ arvoa, osuman todennäköisyys kasvaa hyvin suureksi.

Samalla periaatteella, olkoon kaksi joukkoa syötteitä S ja T. Jos lasketaan $h(s)$ noin $\sqrt{N}$ määrällä satunnaisesti valituilla alkioilla $s \in S$ ja $h(t)$ noin $\sqrt{N}$ määrällä satunnaisesti valituilla alkioilla $t \in T$, ja oletetaan jonkun $h(s)$ olevan arvoltaan sama kuin jokin $h(t)$.

Jos hash-funktion tuotos on noin $n = 60$ bittiä, yllä kuvatulla hyökkäyksellä on suuri mahdollisuus onnistua. On välttämätöntä luoda listat, joiden pituus on noin $2^{n/2} = 2^{30} \approx 10^9$ ja säilöä ne. Tämä on mahdollista useimmilla tietokoneilla. Toisaalta, jos hash-funktion tuotos onkin 128 bittiä, listojen tulee olla noin $2^{64} \approx 10^{19}$ pitkiä. Tämä pituus on liian suuri sekä muistin käytön että ajan käytön näkökulmasta.

3.4 Diskreetti logaritmi

Olkoon p erittäin suuri alkuluku ja halutaan määrittää $L_\alpha(\beta)$. Toisin sanoen, halutaan ratkaista kongruenssiyhtälö $\alpha^x \equiv \beta \pmod{p}$. Tämä voidaan ratkaista suurella todennäköisyydellä syntymäpäivä hyökkäyksellä.

Tehdään kaksi listaa, joiden molempien pituus on $\sqrt{p}$:

1. Ensimmäinen lista sisältää lukuja, jotka ovat muotoa $\alpha^k \pmod{p}$. Luvut on generoitu satunnaisilla luvuilla k, joita on noin $\sqrt{p}$ kappaletta.

2. Toinen lista sisältää lukuja, jotka ovat muotoa $\beta\alpha^{-l} \pmod{p}$. Luvut on generoitu satunnaisilla luvuilla l, joita on noin $\sqrt{p}$ kappaletta.

On todennäköistä, että näistä listoista löytyy ainakin yksi sama luku. Jos näin on, niin

$$\alpha^k \equiv \beta\alpha^{-l}, \text{mistä seuraa } \alpha^{k+l} \equiv \beta (\text{mod } p).$$

Näin saadaan muodostettua yhtälö $x \equiv k + l \pmod{p-1}$, joka on haluttu diskreetti logaritmi. Nyt l ja k ovat tunnettuja, joten yhtälö on ratkaistavissa.

3.5 Useiden törmäyksien hyökkäykset

Tässä kappaleessa havainnollistetaan, miten useimpien hash-funktioiden iteratiivinen luonne tekee niistä oletettua vähemmän resistenttejä useiden törmäyksien hyökkäyksille. On siis olemassa syötteet $x_1, ..., x_n$, joilla on kaikilla sama hash-arvo.

Olkoot r henkilöä ja N mahdollista syntymäpäivää. Pystytään todistamaan, että jos $r \approx N^{(k-1)/k}$, niin on hyvin todennäköistä, että ainakin k:lla henkilöllä on sama syntymäpäivä. Toisin sanoen, odotetaan k:ta törmäystä. Jos hash-funktion tuotos on satunnainen, tämä arvio pysyisi voimassa ja hash-funktion arvoilla olisi k törmäystä. Nimittäin, jos hash-funktion tuotos on n bittinen, $N = 2^n$ mahdollista hash-arvoa, ja jos lasketaan $r = 2^{n(k-1)/k}$ arvoa hash-funktiolle, odotetaan k:ta törmäystä. Seuraavaksi näytetään, että voimme saavuttaa törmäyksiä paljon helpommin.

Monissa hash-funktioissa, esimerkiksi SHA-1, on tiivistefunktio f, joka operoi tietyn mittaisilla syötteillä. On olemassa myös määrätty alkuarvo IV. Käsiteltävä viesti on jatkettu halutun pituiseksi ja sille on suoritettu seuraavat vaiheet:

1. Pilkotaan viesti M lohkoihin $M_1, M_2, ..., M_l$.
2. Olkoon H_0 alkuarvo IV.
3. For $i = 1, 2, ..., l$, olkoon $H_i = f(H_{i-1}, M_i)$.
4. Olkoon $H(M) = H_l$.

SHA-1 tiivistefunktiossa jokaisessa iteraatiossa otetaan 160 bittinen syöte $A||B||C||D||E$ edellisestä iteraatiosta viestilohkon m_i mukana, joka on 512 bittiä, ja tuotoksena saadaan uusi $A||B||C||D||E$, jonka pituus on 160 bittiä.

Oletetaan, että funktion f ja samalla hash-funktion H tuotos on n bittinen. Syntymäpäivähyökkäys voi löytää, noin $2^{n/2}$ askeleella, kaksi lohkoa

m_0 ja m'_0 siten, että $f(H_0, m_0) = f(H_0, m'_0)$. Olkoon $h_1 = f(H_0, m_0)$. Toinen syntymäpäivähyökkäys löytää lohkot m_1 ja m'_1, joille pätee $f(H_1, m_1) = f(H_1, m'_1)$. Jatketaan tällä tavalla. Olkoon

$$h_i = f(h_{i-1}, m_{i-1})$$

ja käytetään syntymäpäivähyökkäystä löytääksemme m_i ja m'_i, joille pätee

$$f(h_i, m_i) = f(h_i, m'_i).$$

Tätä jatketaan niin kauan, kunnes on löytynyt t paria lohkoja
$m_0, m'_0, m_1, m'_1, ..., m_{t-1}, m'_{t-1}$, missä t on myöhemmin määriteltävä kokonaisluku.

Jokaisella 2^t viestistä

$$\begin{gathered}
m_0||m_1||...||m_{t-1} \\
m'_0||m_1||...||m_{t-1} \\
m_0||m'_1||...||m_{t-1} \\
m'_0||m'_1||...||m_{t-1} \\
..... \\
m'_0||m_1||...||m'_{t-1} \\
m_0||m'_1||...||m'_{t-1} \\
m'_0||m'_1||...||m'_{t-1}
\end{gathered}$$

(kaikki mahdolliset kombinaatiot m_i ja m'_i) on sama hash-arvo. Tämä johtuu hash-algoritmin iteratiivisesta luonteesta. Jokaisella laskukerralla $h_i = f(m, h_{i-1})$, saadaan sama arvo h_i riippumatta siitä, että oliko $m = m_{i-1}$ vai $m = m'_{i-1}$. Tämän takia funktion f tuotos on jokaisessa algoritmin vaiheessa riippumaton siitä, että käytetäänkö syötteenä m_{i-1} vai m'_{i-1}. Tästä johtuen lopullinen tuotos on sama kaikille yllä kuvatuille viesteille. Täten meillä on 2^t törmäystä.

Tämä menettely vaatii noin $t2^{n/2}$ vaihetta ja sen odotettu kestoaika on vakio kertaa $tn2^{n/2}$. Olkoon esimerkiksi $t = 2$. Tällöin neljän viestin löytäminen, joilla on sama hash-arvo, kestää noin kaksi kertaa pidempään kuin kahden viestin löytäminen, joilla on sama hash-arvo. Jos hash-funktion tuotos on todella satunnainen, eikä käytetä esimerkiksi iteratiivista algoritmia, tällöin edellä esitelty menetelmä ei toimisi. Oletettu aika, joka kuluisi neljän viestin, joilla on sama hash-arvo, löytämiseen, olisi noin $2^{3n/4}$. Tämä on paljon pidempi kuin aika, joka kuluisi kahden törmäävän viestin löytämiseen. Tämän takia on helpompi löytää törmäyksiä iteratiivisesta hash-algoritmista.

3.6 Hyödyntäminen allekirjoituksessa

Henkilö A on allekirjoittamassa dokumenttia sähköisesti käyttäen erästä allekirjoitus järjestelmää, jossa allekirjoitus muodostetaan dokumentin tiivisteestä. Oletetaan, että hash-funktio tuottaa 60 bittisen tiivisteen. Henkilöä A huolestuttaa, että henkilö B yrittää huijata hänet allekirjoittamaan jonkin muun kuin kyseessä olevan dokumentin. Henkilö A kuitenkin helpottuu ymmärtäessään, että todennäköisyys sille, että petollisella sopimuksella olisi sama tiiviste kuin alkuperäisellä sopimuksella, on $1/2^{60} \approx 1/10^{18}$. Henkilö B voi yrittää useita petollisia sopimuksia, mutta on hyvin epätodennäköistä, että hän löytäisi sellaisen, josta muodostuu haluttu tiiviste. Henkilö B on kuitenkin tutkinut syntymäpäivä ongelmaa ja toimii seuraavasti. Hän löytää sopimuksesta 35 kohtaa, joihin hän voi tehdä pienen muutoksen: lisäämällä välilyönnin rivin loppuun, tekemällä sanamuotoihin pientä muutosta ja niin edelleen. Jokaisessa kohdassa hänellä on kaksi mahdollisuutta: Tehdä pieni muutos tai jättää alkuperäinen muoto. Tästä johtuen hän voi tuottaa 2^{35} erilaista dokumenttia, jotka ovat olennaisesti identtisiä alkuperäisen dokumentin kanssa. Henkilö A tuskin kieltäytyisi mistään näistä sopimuksen versioista. Henkilö B laskee jokaisen 2^{35} sopimuksen tiivisteen ja säilöö ne. Samoin hän tekee 2^{35} versiolle petollisesta dokumentista ja säilöö niiden tiivisteet. Huomioidaan yleinen syntymäpäiväongelma, jossa $r = 2^{35}$ ja $n = 2^{60}$. Nyt $r = \sqrt{\lambda n}$, jossa $\lambda = (2^{35})^2/2^{60} = 2^{10} = 1024$. Tämän takia todennäköisyys sille, että jollakin petollisella dokumentilla olisi sama tiiviste kuin jollakin alkuperäisen kaltaisista dokumenteista, on noin $1 - e^{-1024} \approx 1$. Henkilö B löytää osuman ja pyytää henkilöä A allekirjoittamaan alkuperäisen kaltaisen dokumentin. Henkilö B suunnittelee käyttävänsä henkilön A allekirjoitusta petolliseen dokumenttiin, koska niiden tiivisteet ovat samat. Henkilö A kuitenkin pyytää poistamaan pilkun yhdestä dokumentin lauseesta. Tämän jälkeen hän allekirjoittaa dokumentin, jolla on täysin erilainen tiiviste, kuin dokumentilla ennen tätä pientä muutosta. Henkilön B suunnitelma epäonnistuu. Hänellä on enää mahdollisuus yrittää kehitellä petollinen dokumentti, jolla on täsmälleen sama tiiviste kuin allekirjoitetulla dokumentilla. Tämä on kuitenkin pääasiallisesti mahdotonta.

Tässä henkilö B yrittä käyttää hyödykseen syntymäpäivähyökkäystä. Tässä käytännöllinen oivallus on, että aina pitäsi käyttää hash-funktiota, jonka tiiviste on kaksi kertaa pidempi kuin mitä pidetään tarpeellisena, koska syntymäpäivähyökkäys tehokkaasti puolittaa bittien määrän. Henkilö A toimi suositeltavalla tavalla allekirjoitustilanteessa tekemällä pienen muutoksen dokumenttin. Näin hän mitätöi syntymäpäivähyökkäyksen.

Lähdeluettelo

[1] W. Trappen & L. Washington: *Introduction to Cryptography with Coding Theory*. Pearson Prentice Hall, Upper Saddle River, N.J , 2005.

[2] https://fi.wikipedia.org/wiki/SHA